

Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной
математики

Кафедра вычислительной математики и программирования

Курсовой проект по курсу «Операционные системы»

Реализация взаимодействия между процессами по принципу «Клиенты -
сервер».

Студент: Кварацхелия А. Ю.
Преподаватель: Миронов Е. С.
Группа: М8О-201Б-21
Дата:
Оценка:
Подпись:

Москва, 2023

Условие

Реализовать программы клиента и сервера, в которых клиент отправляет запросы на сервер, а сервер обрабатывает эти запросы, клиент и сервер связаны очередью сообщений.

Метод решения

У клиента есть несколько опций: (регистрируется он в обязательном порядке) создать комнату, присоединиться к комнате, вывести статистику игрока, начать игру.

Код программы

zmq-utils.hpp

```
#ifndef ZMQ_UTILS_HPP
#define ZMQ_UTILS_HPP
```

```
#include <string>
#include <iostream>
#include <zmq.hpp>
```

```
namespace zus
```

```
{
```

```
    class Socket
```

```
    {
```

```
        public:
```

```
            Socket(const std::string &address, const std::string &socketType)
```

```
            void SendMessage(const std::string &requestStr);
```

```
            std::string RecieveMessage();
```

```
            std::string SendRequest(const std::string &requestStr);
```

```
            std::string SendResponse(const std::string &responseStr);
```

```
            ~Socket();
```

```
        private:
```

```
            std::string address;
```

```
            std::string socketType;
```

```
            zmq::context_t context;
```

```
            zmq::socket_t socket;
```

```

        struct Private;
    };
}

#endif

server-utils.hpp

#ifndef SERVER_UTILS_HPP
#define SERVER_UTILS_HPP

#include <string>

int Unpack(std::string &str);

#endif

utils.hpp

#ifndef UTILS_HPP
#define UTILS_HPP

#include <string>
#include <regex>
#include <set>

class RoomName
{
    public:
        RoomName();
        RoomName(std::string &name);

        std::string data;
        std::string formatted;

        ~RoomName();
    private:
        ;
};

std::istream &operator >> (std::istream &in, RoomName &obj);

```

```

class Room
{
    public:
        RoomName name;
        std::string port;
        bool running{0};

        Room(std::string &line);
        Room(std::string &name, std::string &port);
        Room(std::string &name, std::string &port, bool running);
        ~Room();

    private:
};

```

```

class RoomsManager
{
    public:
        RoomsManager();

        bool Member(std::string &name);
        std::string GetPort(std::string &name);
        std::string AddRoom(std::string &name);
        void StartRoom(std::string &name);

        ~RoomsManager();
    private:
        std::set<Room> rooms;
};

```

#endif

zmq-utils.cpp

#include "zmq_utils.hpp"

```

#include "allias.hpp"
#include <exception>
#include <zmq.hpp>

struct zus::Socket::Private
{
    static void SendMessage(zus::Socket &self, const std::string &requestStr)
    {
        try
        {
            zmq::message_t response(requestStr.data(), requestStr.length());
            zmq::send_result_t responseStatus = self.socket.send(response, zmq::
        }
        catch(std::exception &exc){
            std::cerr << "[socket:_" << self.address << "]" << exc.what() <<
        }
    }

    static std::string RecieveMessage(zus::Socket &self)
    {
        std::string result;

        try
        {
            zmq::message_t request;
            zmq::recv_result_t requestStatus = self.socket.recv(request, zmq:
            result = request.to_string();
        }
        catch(std::exception &exc){
            std::cerr << exc.what() << std::endl;
        }

        return result;
    }
};

```

```

zus::Socket::Socket(const std::string &address, const std::string &socketType
{

```

```

this->socketType = socketType;
this->address = address;

if (!socketType.compare(utl::SERVER)){
    this->socket = zmq::socket_t(this->context, zmq::socket_type::rep);
}

if (!socketType.compare(utl::CLIENT)){
    this->socket = zmq::socket_t(this->context, zmq::socket_type::req);
}

try
{
    if (!this->socketType.compare(utl::SERVER)){
        this->socket.bind(address);
    }

    if (!this->socketType.compare(utl::CLIENT)){
        this->socket.connect(address);
    }
}
catch (std::exception &exc){
    //std::cerr << sym::RED << "[Error:]" << sym::RESET |
    //    << " socket: " << address << ": " |
    //    << exc.what() << std::endl;
}
}

void zus::Socket::SendMessage(const std::string &requestStr)
{
    try
    {
        zmq::message_t response(requestStr.data(), requestStr.length());
        zmq::send_result_t responseStatus = this->socket.send(response, zmq::
    }
    catch(std::exception &exc){
        std::cerr << "[socket:_msg:_]" << ",_" << this->address << "]" << exc
    }
}

std::string zus::Socket::RecieveMessage()
{

```

```

    std::string result;

    try
    {
        zmq::message_t request;
        zmq::recv_result_t requestStatus = this->socket.recv(request, zmq::recv_flags::none);
        result = request.to_string();
    }
    catch(std::exception &exc){
        std::cerr << exc.what() << std::endl;
    }

    return result;
}

std::string zus::Socket::SendRequest(const std::string &requestStr)
{
    //Private::SendMessage(*this, requestStr);
    //return Private::ReceiveMessage(*this);
    SendMessage(requestStr);
    std::string responseStr = ReceiveMessage();

    return responseStr;
}

std::string zus::Socket::SendResponse(const std::string &responseStr)
{
    std::string requestStr = ReceiveMessage();
    SendMessage(responseStr);

    return responseStr;
}

zus::Socket::~~Socket()
{
    if (!this->socketType.compare(utl::CLIENT))
    {
        std::string request = utl::TERMINATOR;
        SendRequest(request);
        this->socket.disconnect(this->address);
    }
}

```

```

        this->socket.close();
        this->context.close();
    }

```

client.cpp

```

#include "zmq_utils.hpp"
#include "utils.hpp"
#include "alias.hpp"

```

```

#include <cstdlib>
#include <iostream>
#include <iterator>
#include <string>
#include <regex>
#include <zmq.hpp>

```

```

std::string WithId(std::string &id, std::string &str){
    return std::string(id + "_" + str);
}

```

```

int main(int argc, char* argv[])
{

```

```

    std::string port = argv[1];
    std::string id = argv[2];

```

```

    std::string address = "tcp://localhost:" + port;
    zus::Socket socket(address, utl::CLIENT);

```

```

    ClientCommand cmd(id);

```

```

    std::string input, response, request = cmd.INIT_CLIENT;

```

```

    std::cout << "Log_in->";
    std::cin >> input;
    request += "_" + input + "_";

```

```

    response = socket.SendRequest(request);
    request = id + "_";

```

```

    std::cout << "\nCREATE_-create_room,\nCONNECT_-connect_to_the_room.\n";
    std::cin >> input;

```



```

request += input;

std::cout << "\nEnter_lobby_name->";
std::cin >> input;
request += "_" + input;

std::cout << request << '\n';
response = socket.SendRequest(request);

std::cout << "\nCommands:\n0--start_game;\n" \
    << "1<number>--choose<number>\n" \
    << "2<user>--print_stat_of<user>\n";

while (std::getline(std::cin, input))
{
    if (input.length())
    {
        if (input.compare(msg::START_GAME)){
            execl("./gamer", "gamer", argv[1], "5555");
        }

        request = id + "_" + input;
        response = socket.SendRequest(request);
    }

    if (response.length()){
        std::cout << response << '\n';
    }
}

request = cmd.TERMINATOR;
socket.SendRequest(request);

return 0;
}

```

server.cpp

```

#include "utils.hpp"
#include "zmq_utils.hpp"
#include "alias.hpp"
#include "server_utils.hpp"

```

```

#include <cstdio>
#include <fstream>
#include <random>
#include <sstream>
#include <string>
#include <utility>
#include <vector>
#include <zmq.hpp>
#include <unistd.h>
#include <queue>
#include <set>

using pii = std::pair<int, int>;

class StatChecker
{
    private:

    public:
        std::map<std::string, int> data;
        StatChecker()
        {
            std::ifstream fin(room::PATH_TO_USERS_DB);

            std::string name;
            fin >> name;

            int record;
            fin >> record;

            data[name] = record;
        }
        bool Member(std::string &key){
            return data.count(key);
        }
        ~StatChecker()
        {
            std::ofstream fout(room::PATH_TO_USERS_DB);

            for (std::map<std::string, int>::iterator it = data.begin(); it != data.end(); ++it)
                fout << it->first << "_" << it->second << '\n';
        }

```

```

        }
    }
} stat;

struct TArgs
{
    int id;
    std::string command;
    std::string arg;
};

TArgs GetCommand(std::string &str)
{
    TArgs result;

    std::stringstream strm;
    strm << str;
    std::string str1;

    std::getline(strm, str1, '_');
    result.id = std::atoi(str.data());
    std::getline(strm, str1, '_');
    result.command = str1;

    if (!str1.compare(msg::KILL)){
        std::getline(strm, str1);
    }
    else {
        std::getline(strm, str1, '_');
    }
    result.arg = str1;

    return result;
}

static std::set<std::string> logged;

std::string RunCommand(std::string &srcStr)
{
    std::string result;
    TArgs cmd = GetCommand(srcStr);

```

```

RoomsManager manager;

if (!cmd.command.compare(msg::INIT_CLIENT))
{
    if (!logged.count(cmd.arg)){
        result = msg::NO_SUCH_USER;
    }
    else
    {
        logged.insert(cmd.arg);
        result = msg::SUCCES;
    }
}

if (!cmd.command.compare(msg::CREATE))
{
    if (manager.Member(cmd.arg)){
        result = msg::ROOM_EXISTS;
    }
    else {
        result = manager.AddRoom(cmd.arg);
    }
}

if (!cmd.command.compare(msg::CONNECT))
{
    if (!manager.Member(cmd.arg)){
        result = msg::ROOM_NOT_FOUND;
    }
    else {
        result = manager.GetPort(cmd.arg);
    }
}

if (!cmd.command.compare(msg::START_GAME))
{
    int pid = fork();

    if (!pid){
        execl("./room", "room", "5555", NULL);
    }
}

```

```

if (!cmd.command.compare(msg::STAT))
{
    if (!stat.Member(cmd.arg)){
        result = "0";
    }
    else
    {
        std::map<std::string , int>::iterator it = stat.data.find(cmd.arg)
        result = std::to_string(it->second);
    }
}

return result;
}

int main(int argc , char* argv[])
{
    std::string port = argv[1];
    std::string address = "tcp://*:" + port;
    zus::Socket socket(address , utl::SERVER);

    int numOfClients = 2;

    while (true)
    {
        std::string requestStr = socket.RecieveMessage();

        if (!requestStr.compare(utl::TERMINATOR))
        {
            —numOfClients;
            socket.SendMessage(requestStr);
            if (numOfClients < 1){
                break;
            }
        }
        else {
            socket.SendMessage(RunCommand(requestStr));
        }
    }
}

```

```
        return 0;  
    }
```

Выводы

В процессе выполнения лабораторной работы были приобретены навыки практического применения и создания классов и «статиков», также были расширены знания в области использования сокетов очередей сообщений.